Res. Lett. Inf. Math. Sci., (2001) **2**, 11–17
Available online at http://www.massey.ac.nz/~wwims/rlims/

Algebraic Conversions

Bruce Mills
I.I.M.S., Massey University Albany Campus, Auckland, New Zealand
b.i.mills@massey.ac.nz

Abstract
An examination of the pure algebraic properties of computational type conversion leads to a new generalizations of the concept of a homomorphism for which the term *conversion* seems appropriate. While an homomorphism is a mapping that respects the value of all terms, a conversion is a mapping that respects the value of all sufficiently small terms. Such a mapping has practical value, as well as theoretical interest that stems from conversions forming a category. This paper gives a precise definition of the concept and demonstrates an application to formal computer science based on work completed by the author in his PhD thesis.

1 Background

The purpose of this discussion is to introduce a particular generalization of the concept of an homomorphism, and to motivate its use. The new concept originated in a problem of optimization of an automated computation, but is presented here as an abstract entity. The formality of the presentation is constrained in order to keep the intuitive computational meaning of the concept clear. However, sufficient comments and references are given to enable the interested reader to fill in the details of a fully formal approach.

Formally, computational types are, herein, equated with universal algebras, but treated from an abstract algebraic perspective. An universal algebra [1] is an underlying set V of values together with a collection of operations $f_i : V^{a_i} \to V$, each with their own arity, $a_i \in \mathbb{Z}^+$. The positive reals $(\mathbb{R}^+, \times, ^{-1})$ under multiplication and multiplicative inversion are a concrete example. The underlying set of values is $\mathbb{R}^+$, the operations are multiplication which is binary, having arity 2, and inversion which is unary, having arity 1. The definition is very general, and includes as special cases groups, rings, fields, and such-like from abstract algebra, as well as abstract data types, which form a central part of formal code analysis and synthesis in computer science [2].

The characteristic which makes universal algebras particularly suited to formal computer science theory is the concentration on the structure and value of expressions. In this context an expression such as $x \times y$ with unknown quantities x and y is taken as a formal term in the symbols x and y. This is a generalization of the technique [7] for forming the ring of polynomials $F[x]$ over a field F as a ring of formal sums of powers of x, with coefficients in F. In the abstract algebraic approach, the associated evaluation homomorphisms ϕ_i takes terms in a formal language [3] into the set V of values. For example, given a formal term $x_1 \times x_2$, the value of $\phi_{a,b}(x_1 \times x_2)$ where $a, b \in V$ is the particular element $c \in V$ such that $a \times b = c$.

A full discussion of the constructions implied by the above will not be undertaken here, the interested reader is referred to the literature. An intuitive understanding of the concepts is sufficient for the following discussion.

Sometimes we can extend an ability to compute in one algebraic structure into an ability to compute in another. Continuing the example of real arithmetic above, it is well known that computation of a product $a \times b$ can proceed by the method of logarithms. $a \times b = \exp(\log a + \log b)$. A more complex expression such as $a \times b \times c^{-1}$ can be mapped in a similar manner to $\exp(\log a + \log b + (-\log c))$. In general we may wish to evaluate an arbitrary finite term in the formal algebra of expressions over $(\mathbb{R}^+, \times, ^{-1})$. The method for doing this by logarithms is apparent, and easily converted into instructions for automated computation.

This computational transformation may be justified in this case by the observation that that log is an isomorphism of $(\mathbb{R}^+, \times, ^{-1})$ onto $(\mathbb{R}, +, -)$, the algebra of the reals under addition and additive negation. The former algebra could be represented, in a digital computer, by storing it as the latter, and providing the logarithm and exponential functions as input and output conditioning, solely for the purpose of interacting with the external environment. Such an approach is standard both in pure algebra and in the theory of computation.

But, in contemporary digital computers, pure algebras are often represented by data structures that are not isomorphic to the pure version. Integer arithmetic $(\mathbb{Z}, +, \times)$ is typically represented by $(\mathbb{Z}_n, +, \times)$, where n is some power of 2, and real arithmetic $(\mathbb{R}, +, \times)$ is represented by floating point arithmetic, which is not even associative [9]. Such a state of affairs, although historically and technologically precedented, has driven a wedge between pure mathematics and practical computing.

One approach for attempting reconciliation of pure mathematics and the pragmatic art of automated computation is to build correct models of the pure algebraic structures [5]. While this approach is viable and even indispensable in many cases it also suffers from a lack of efficiency [6]. It is the author's personal feeling on this matter that a more complete reconciliation will only be obtained by also extending the mathematics to incorporate the actual behaviour of the efficient computational types, if only so that pure and efficient structures can be designed from the ground up.

As part of this process, this discussion is designed to show not simply *that* but *why* an homomorphism is not strictly required, and at the same time to define a pure algebraic concept which could be used as a tool to study these types of non-homomorphic representation.

2 Motivation

The triggering motivation for the construction presented here was a question tackled by the author [8] about the optimality of certain special case matrix algorithms. Rather than simply re-arranging the required operations as a bookkeeping exercise, the task being examined was the reparameterisation of a matrix universal algebra in order to allow computation via non equivalent expressions.

For example, if we express symmetric second order real matrices as the image of a map from $\mathbb{R}^2 \to \mathbb{R}^{2,2}$ as follows:

$$(a, b) \to \begin{bmatrix} a & b \\ b & a \end{bmatrix},$$

then we can compute with the matrices by computing with pairs of numbers. In particular $(a, b) * (c, d) = (ac+bd, bc+ad)$ is the required formula for multiplying matrices thus parameterised. Such a parameterisation is a mild improvement over the naive storage of all four matrix elements. However, it is also simply the removal of a fairly obvious redundancy, a simple bookkeeping exercise.

None-the-less, is this expression optimal? In this parameterisation it is. But, a different, less obvious, parameterisation gives further improvement. By storing the matrices as

$$(a, b) \to \begin{bmatrix} (a+b)/2 & (a-b)/2 \\ (a-b)/2 & (a+b)/2 \end{bmatrix},$$

matrix multiplication is computed by the simple expression: $(a, b) * (c, d) = (ac, bd)$. Which has three times less operations, and reveals an hidden independence of the dimensions which could be used to compute the result in parallel.

Is *this* expression optimal? The problem in answering this question, which at first glance may have appeared trivial, is that the optimal expression depends on the parameterisation. Perhaps some space filling curve gives a parameterisation of $\mathbb{R}^2$ by $\mathbb{R}$ which would allow the operation to be computed by, say, a single multiplication. From a pragmatic perspective, in the computational world of floating point representation, the required approximation to such parameterisations can

be quite easy to compute, and from the pure perspective, the desire is to optimize over *all* possible parameterisations, not just, for example, continuous ones.

The following discussion demonstrates that the obvious answer is right, no such parameterisation is possible. But, it is not nearly so *obvious* as it might at first seem. In a very real sense the assertion is only just true. Even though the desired parameterisation does not exist, there is a sequence of parameterisations whose behaviour limits to the desired behaviour[1].

3 Definition and properties

We now simplify our real arithmetic example to concentrate only on the operations of addition $(\mathbb{R}, +)$ and multiplication $(\mathbb{R}^+, \times)$. The isomorphism, log, has the definitive property that $\log(a \times b) = \log(a) + \log(b)$. However, computationally, the important thing is the existence of the function, exp, with the property that $a \times b = \exp(\log(a) + \log(b))$. Without the ability to map the result back again to the original domain, the conversion would be of limited use. In the logarithm case *exp* happens to be the inverse of log, however, the universal truth of the latter equality does not imply that this must be so.

As an example, mapping $\mathbb{Z}_n \to \mathbb{Z}$ by mapping $1 \to 1$ and $2 \to 2$ and so on, means that we can evaluate expressions in $(\mathbb{Z}_n, +)$, by mapping into $(\mathbb{Z}, +)$ and then mapping back. In this case the map from $\mathbb{Z}$ to $\mathbb{Z}_n$ *is* a homomorphism but not an injective one, and is not the inverse of the forward map.

The pragmatic case allows the use of two maps, not specifically the inverses of one another. The first turns a problem in domain A into a problem in domain B, and the second maps an answer in domain B back into domain A. In computational terms, we have converted the problem in domain A to one in domain B. The generic idea of such a reduction [4] is a central concept in the theory of computational complexity. But, the emphasis of this paper is on the mathematical properties of certain pairs of mapping between algebraic structures, which do not simply re-arrange the problem, but change its overt nature completely.

Definition
A pair (f, g) of maps $f : A \to B$ and $g : B \to A$, is a conversion of $(A, *)$ into $(B, \circ)$, if $\forall a, b \in A$, $g(f(a) \circ f(b)) = a * b$.

One way in which this can occur is that $g : B \to A$ is an onto homomorphism and f is any one sided inverse of g, such that $f(a) \in g^{-1}(a)$ and thus $g(f(a)) = a$. But, this is not the only way.

An example, less trivial than that presented above emphasizes the point.

Given a non negative integer a, and a positive integer b,

define $a\%b = a - b * \lfloor a/b \rfloor$ so $a\%b$ is positive, and in the discrete interval $[0 \,..\, b-1]$ and $a = b(\lfloor a/b \rfloor) + a\%b$

define $bind_n(a, b) = n^4 * bind_n(\lfloor a/n \rfloor, \lfloor b/n \rfloor) + n^2(a\%n) + b\%n$
with $bind_n(0, 0) = 0$

define $free_n(t) = (\lfloor t/n^2 \rfloor \%(n^2), t\%(n^2)) + free_n(\lfloor t/n^4 \rfloor)$
with $free_n(0) = 0$

Theorem 1 $free_n(bind_n(a_1, b_1) + bind_n(a_2, b_2)) = (a_1 + a_2, b_1 + b_2)$.

Proof: Use induction, with hypothesis that the assertion is true for $0 < a_1, a_2, b_1, b_2 < n^m$, base case is $m = 0$ is immediate. We prove the case $m = 1$, If $0 < a_1, a_1, b_1, b_2 < n$, then $\lfloor a/n \rfloor, \lfloor b/n \rfloor = 0$, $bind_n(0, 0) = n^4 * bind_n(0, 0) + 0 = 0$ So, $bind(a_1, b_1) = n^2(a_1\%n) + b\%n$, similarly for a_2 and b_3. So the expression is $free_n(n^2(a_1\%n^2) + b_1\%n^2 + n^2(a_2\%n) + b_2\%n^2)$ which is, $free_n(n^2((a_1 + a_2)\%n^2) + (b_1 + b_2)\%n) = ((a_1 + a_2)\%n^2, (b_1 + b_2)\%n^2)$. since all the numbers are in $[0..n]$, we see that this is $(a_1 + a_2, b_1 + b_2)$, a similar argument demonstrates the general inductive case. ■

[1] Of course the parameterisations do not limit to a specific parameterisation

But it is fairly clear that typically neither *bind* nor $free$ will be an homomorphism. In fact, there are no homomorphisms from $(\mathbb{R}^2, +)$ into $(\mathbb{R}, +)$.

Note the similarity and distinction between homomorphism and conversion. Homomorphism is characterized by $\phi(a * b) = \phi(a) \circ \phi(b)$, while a conversion requires $a * b = g(f(a) \circ f(b))$. In the case of any onto homomorphism we can find an operation ψ such that $\phi(\psi(a)) = a$, and thus $\phi(\psi(a) \circ \psi(b)) = \phi(\psi(a)) * \phi(\psi(b)) = a * b$. So, each homomorphism leads to a collection of related conversions. However, the shift of ψ to the other side of the equality has a profound effect. While for a homomorphism for any integer $n > 0$ we have $\phi(\Pi_{i=1}^n a_i) = \Pi_{i=1}^n \phi(a_i)$, this is not always the case for a conversion.

Direct computation shows that

$$free_2(1 \times bind_2(1,1)) = (1,1)$$
$$free_2(2 \times bind_2(1,1)) = (2,2)$$
$$free_2(3 \times bind_2(1,1)) = (3,3)$$
$$free_2(4 \times bind_2(1,1)) = (1,2)$$

A direct computation also shows that ...

$$free_2(n^2 \times bind_n(1,1)) = (1, n^2),$$

but, for $\alpha \in [0 \,..\, n^2 - 1]$,

$$free_2(\alpha \times bind_n(1,1)) = (\alpha, \alpha),$$

This suggests the following theorem:

Theorem 2 $free_n(\Sigma_{i=1}^m bind_n(a_i, b_i)) = (\Sigma_{i=1}^m a_i, \Sigma_{i=1}^m b_i)$ *exactly when* $\Sigma_{i=1}^m((a_i n^{p-1})\%n^p), \Sigma_{i=1}^m((b_i n^{p-1})\%n^p < n^2$ *for each non negative integer* p.

Proof: Looking at the proof of Theorem 1 we see that the essential point is that in the evaluation of $free_n(n^2((a_1 + a_2)\%n^2) + (b_1 + b_2)\%n)$ to a value of $((a_1 + a_2)\%n^2, (b_1 + b_2)\%n^2)$ the requirement is that the numbers are in $[0 \,..\, n]$ and so this is $(a_1 + a_2, b_1 + b_2)$. It would not matter how many summands where involved, as long as this condition holds. Further, if the condition does not hold it is apparent that the required equality does not hold either. Thus, the theorem follows. ■

As a direct corollary, it is clear that any summation with no more than n summands will not have the auxiliary sums from the theorem greater than n^2, and thus can be correctly computed by converting via $bind_n$, summing, and then converting back via $free_n$.

The same result can be obtained for the reals by defining

$$rbind_n(a,b) = \lim_{m\to\infty} n^{-m} bind_n(\lfloor an^m \rfloor, \lfloor bn^m \rfloor)$$
$$rfree_n(a) = \lim_{m\to\infty} n^{-m} free_n(\lfloor an^m \rfloor)$$

and following through the basic proof for the integer case checking that the limit is not affected.

Thus, and most importantly, it has been shown that for each $n \in \mathbb{Z}^+$ there is a conversion from $(\mathbb{R}^2, +)$ to $(\mathbb{R}, +)$ which can be used to evaluate a summation of up to n summands. As long as we know beforehand how many summands there are, we can correctly compute the value of the sum of a number of pairs of numbers by converting it into a sum of single numbers.

It now remains to answer the question of whether there is exists a single conversion that will compute all (or perhaps just all finite) summations correctly. The is the subject of the rest of the discussion.

4 The duration of a conversion

We now formalise the observation made at the end of the previous section. The definitions will be given for the case of an universal algebra with a single binary operation, however the construction of the appropriate definition for universal algebras with more operations, with different arities, should be clear from the discussion. Conceptually the following definition is fairly straight forward, but requires a bit of careful wording so that infinite expressions can be included.

4.1 The general definition

Given a set S of symbols, otherwise undefined, a formal S-expression is a rooted, leaf S-weighted binary tree. If t_1 and t_2 are S-expressions, we define $t_1 \# t_2$ to be the S-expression formed by adding a new root node whose left and right children are the root nodes of t_1 and t_2 respectively. If a collection E of S-expressions is closed under $\#$ then $(E, \#)$ is an universal algebra. The set of weights of single leaf trees in E is called the expression base of $(E, \#)$. By the *size* of an expression is meant the number (possibly infinite) of leaf nodes.

Let $(A, *)$ be universal algebra, and $(E, \#)$ be an universal algebra of A-expressions such that the expression base of $(E, \#)$ is A. For each $a \in A$ let $t(a)$ be the single leaf tree weighted by a. An evaluation of $(E, \#)$ in A is a homomorphism $v : (E, \#) \to (A, *)$ such that $v(t(a)) = a$.

Let v_A be an evaluation of E in A. Given another universal algebra $(B, \circ)$, and a map $f : A \to B$, the collection of expressions $f(E)$ is the set of expressions obtained by changing each weight a to the weight $f(a)$. Let v_B be an evaluation of $f(E)$ in B. Let $g : B \to A$ such that (f, g) is a conversion of A into B.

If there exists a cardinality c, such that for each expression $e \in E$ that is at most size c, $g(v_B(f(e))) = v_A(e)$ then the conversion (f, g) is said to be of duration at least c. If the equality holds for all $e \in E$, then the conversion is said to be permanent in E.

4.2 Application to topological abelian groups

We now specialise the definition somewhat and derive results that will enable us to answer directly the question left open at the end of section 3.

If $(A, *)$ is a topological universal algebra with an identity element, then we can usefully define a value for some countably infinite expressions by a form of limit. Specifically, for any infinite expression generate the set of all finite expressions obtained by replacing enough subtrees by the identity element in $(A, *)$. If the set of values of these finite expressions has a unique accumulation point, then we call it the limiting value of the infinite expression, and take it as the value of the infinite expression. In the following we assume that we are using such an evaluation.

Let $(A, *)$ be an abelian group. It should be clear from the above discussion that any evaluation is uniquely defined on the finite expressions, and that any evaluation that respects limits must be uniquely defined on countable sized expressions which have a limit. Such an evaluation corresponds to the intuitive notion of evaluation of expression in an universal algebra.

An infinite series in an abelian group is an expression in which the left hand child of each node is a leaf. Thus there is exactly one element at any given depth i from the root node. Call this the element a_i. The value of such an expression (if it exists) is denoted by $\Sigma_{i=1}^{\infty} a_i$.

Let $(B, \circ)$ be another topological abelian group, and (f, g) a conversion of A to B. If the sum $\Sigma_{i=1}^{\infty} f(a_i)$ has a limiting value whenever $\Sigma_{i=1}^{\infty} a_i$ does, and further if when it does we also have that $g(\Sigma_{i=1}^{\infty} f(a_i)) = \Sigma_{i=1}^{\infty} a_i$ Then (f, g) is said to be a permanent conversion of A to B.

Theorem 3 *If (f, g) is a permanent conversion, then g is a homomorphism of $G =< f(A) >$ onto A.*

Proof: By definition each a and b in G can be expressed as $a = \Sigma(f(a_i)$ and $b = \Sigma(b_i)$ for some a_i and b_i all in A. so $g(a+b) = g(\Sigma f(a_i) + \Sigma f(b_i)) = g(\Sigma(f(a_i) + f(b_i))) = \Sigma(a_i + b_i) = \Sigma a_i + \Sigma b_i$. ■

We can also demonstrate that g is continuous as follows:

Let $z_i \in A$ be defined for $i \in \mathbb{Z}$, and suppose that $z_i \to z \in A$. Let $\Delta_i = z_{i+1} - z_i$ and $\delta_i = g(\Delta_i)$.

So $z_n = \Sigma_{i=1}^{n} \Delta_i$

Thus $g(z_n) = g(\Sigma_{i=1}^{n} \Delta_i) = \Sigma_{i=1}^{n} \delta_i$.

Let $y_n = \Sigma_{i=1}^{n} \delta_i$ and $y = \Sigma_{i=1}^{\infty} \delta_i$

Now $g(z) = g(\Sigma_{i=1}^{\infty} \Delta_i = \Sigma_{i=1}^{\infty} g(\Delta_i) = \Sigma_{i=1}^{\infty} \delta_i$

So $z_i \to z$ implies that $g(z_i) \to g(z)$.

So $g : G \to A$ is a continuous homomorphism.

Finally, g must be onto A, since any element in A can be obtained as a sum of two other values in A.

We have thus proved

Theorem 4 *If (f, g) is a permanent limit respecting conversion from one topological abelian group to another, then g is an onto homomorphism.*

We are now in a position to demonstrate the answer to the question of optimality of addition in $(\mathbb{R}^2, +)$. The question can be answered in a slightly more general setting. Consider a permanent conversion (f, g) from $(\mathbb{R}^m, +)$ into $(\mathbb{R}^n, +)$. From the above discussion $g : \mathbb{R}^n \to \mathbb{R}^m$ must be a continuous onto homomorphism. But the dimension of $g(B)$ as a vector space is no greater than the dimension of B, so for this to occur, the dimension of A must be no greater than the dimension of B.

Thus $(\mathbb{R}^n, +)$ is optimal, in so far as it cannot be permanently converted to a smaller expression using the standard arithmetic operations (due to the logarithm relation between addition and multiplication).

It was part of the point of this paper to show that this was only just true, ie, we can convert for any duration, as long as we know what that duration is before hand.

It is however interesting to note that the amount of work to add the numbers, give say, a decimal digit representation, is not improved by the conversion, since the number of digits to handle increases.

5 Discussion

The fact that we have decided what operations we can have in $\mathbb{R}$ is important. Since $\mathbb{R}$ and $\mathbb{R}^2$ are equivalent as sets we could just get any one to one map between them, and construct an appropriate operation that mimics two dimensions of addition, but this operation is not available, and if we constructed it out of arithmetic operations we would need at least two of them, which means that there is no saving on the work required.

We can look at this a bit like a local topological group, ie, that the structure is a group as long as you don't multiply things to far from the identity element. In a similar manner, we see a conversion as respecting the structure of the evaluation homomorphism acting on the class of formal expressions in the algebra. As long as you stick to simple expressions the conversion produces the right answer, but outside some boundary it might not.

A similar phenomenon in computing is type conversion from int to float. If we convert and then add, and take the result back to an int then sometimes the result is correct, but if the values are too big, the float will drop significant digits in the result. Another example, modulo n arithmetic converted to modulo m arithmetic, if $m > 2n$ is not a multiple of n, then a single addition will work, but multiple additions might not.

We can compose conversions in a natural manner, by composing the individual functions. Define $(f_1, g_1) \circ (f_2, g_2)$ to be $(f_2 \circ f_1, g_1 \circ g_2)$. Composition of conversions is associative, since composition of functions is. Further, the pair composed of two copies of the identity function in a

single algebraic structure is a conversion from an abstract algebraic structure to itself. Thus, the set of algebraic structures, with conversions, forms a category.

The challenge here is to construct a mature algebraic theory of conversions. Algebraically universal conversions with the projective property, similar to projective modules, could be useful for determining the optimality of whole classes of algorithms by making it possible to demonstrate the non-existence of conversions to simpler structures. Generically, the relationship of conversions to homomorphisms seems similar to that between semi-groups and groups. An explicit connection might be interesting. Construction of a natural group $Con(A, B)$ of conversions from A to B seem problematical due to the interaction between the two halves of the conversion. However, such a construction in similar manner to $Hom(A, B)$, could lead to a structure theory of conversions, including a form of homology theory. Further work will be required to answer these questions.

References

[1] "A course in universal algebra", S.Burris, H.P.Sankapanavar, pub: Springer-Verlag, [1981].

[2] "Specification of Abstract Data Types", J.Loeckx, H.D.EHrich, M.Wolf. pub: Wiley [1999].

[3] "Elements of the theory of computation", H.R.Lewis and C.H.Papadimitriou, pub: Prentice Hall International, [1998]

[4] "Computers and intractability", M.R.Garey, D.S.Johnson, pub: W.H.Freeman [1979].

[5] "Modern computer algebra", Joachim von zur Gathen and Jürgen Gerhard Cambridge University Press, [1999].

[6] Computational geometry abounds with examples. See "Handbook of discrete and computational geometry" edited by JE.Goodman and J.O'Rourke, CRC press 1997.

[7] "The Algebraic Foundations of Mathematics", R.A.Beaumont and R.S.Pierce, pub: Addison Wesley,1963

[8] "Special Case Algorithms", by B.I.Mills, PhD Thesis accepted 1999, The University of Western Australia Depts of Mathematics and Computer Science.

[9] For example, in an explicit test of ANSII C code, using `double` floating point arithmetic, with `x=1e33`, `y=111983274845.6`, `z=10e-33` the following result was computed: `(xy)z - x(yz) = -0.00024414062500000000`.